LABO 12 – TRIS ET COMPLEXITE
Informatique 1

Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

But du laboratoire (4 périodes)

1. Le but de ce laboratoire est de tester la complexité de deux algorithmes de tri différents et de comparer leur temps d'exécution respectifs sur de grandes séries de données. Vous pourrez ainsi vous rendre compte de l'impact de la complexité d'un algorithme sur son temps d'exécution lorsque le nombre de données augmente.
2. La durée estimée pour réaliser ce laboratoire est de **quatre périodes**. Si le temps imparti ne devait pas suffire, vous êtes invités à terminer le travail en dehors des heures de cours.
3. Vous pouvez trouver cette donnée sous forme électronique se trouve sur http://inf1.begincoding.net à la rubrique *Labo 12*.

Partie 1 – Génération de tableaux aléatoires

Dans cette première partie, vous allez créer une classe permettant de générer des tableaux aléatoires qui devront être triés dans le reste de ce laboratoire. Cette classe qui servira à générer différentes sortes de tableaux sera nommée `ArrayFactory`.

Une classe abstraite

Cette classe ne contiendra **que** des méthodes statiques (c'est-à-dire qu'on ne créera pas d'instance de la classe pour utiliser les méthodes). Une telle classe est nommée ***abstraite*** car elle appelée à ne ***jamais être instanciée du tout***.

Pour être certain que la classe ne sera jamais instanciée, vous pouvez ajouter comme attribut à la classe le nouveau mot-clé `abstract`.

Tâche 1

1) Créez un projet *Lab12* et créez une nouvelle classe `ArrayFactory`.
2) Ajoutez une méthode à la classe `ArrayFactory` avec le prototype suivant :

```
static int[] createRandomArray(int size, int maxValue)
```

Le premier paramètre de cette méthode indique combien d'éléments devront être contenus dans le tableau retourné. Le paramètre `maxValue` indique quant à lui la valeur maximale des nombres aléatoires générés dans le tableau.

La classe Random

Pour générer ces nombres, vous allez utiliser la classe `Random` qui est fournie par *Java*. Pour cela, vous allez créer une instance de cette classe puis utiliser la méthode `nextInt(int val)` sur l'objet créé. Cette méthode retourne un nombre entier entre 0 et `val` (non compris). Si vous désirez que la séquence de nombres aléatoires soit toujours la même à chaque fois que vous lancez votre programme, utilisez le constructeur `Random(long val)` en fixant `val` à une valeur quelconque. *Note* : pour utiliser cette classe, écrivez `import java.util.Random;` tout en haut de votre classe. Ceci aura pour effet de rendre visible la classe `Random` que *Java* vous donne pour votre programme.

3) Ajoutez la méthode `static int[] createInvertedSortedArray(int size)` à la classe. Elle permettra de créer un tableau dont les éléments sont les nombres de `size-1` à `0`. Autrement dit, un tableau dont les éléments vont en décroissant.
4) Ajoutez finalement une méthode `static int[] createShuffleArray(int size)`. Cette méthode retourne un tableau dont les éléments sont mélangés comme suit : {0, size – 1, 1, size – 2, …}. On obtient ainsi en fonction des valeurs de `size` :

Size 4: {0, 3, 1, 2}, *Size 5:* {0, 4, 1, 3, 2}, *Size 7 :* {0, 6, 1, 5, 2, 4, 3}

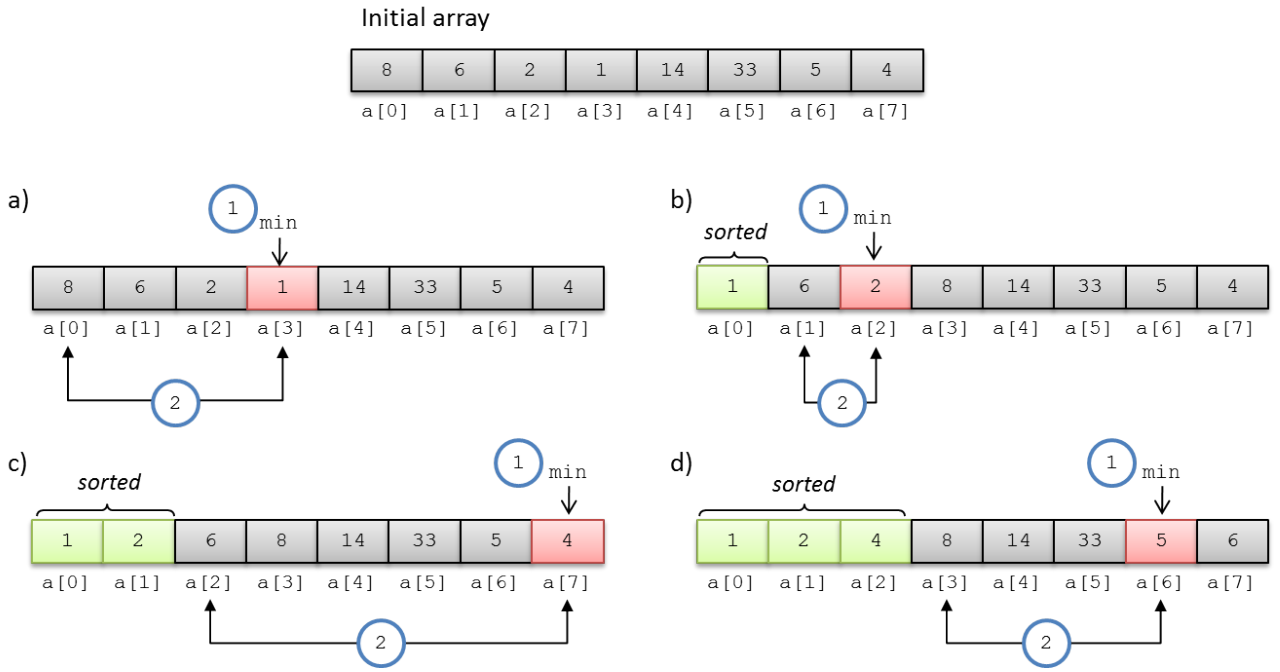
Tâche 2

Vous allez maintenant créer une classe *SortApplication* qui contiendra le *main* de l'application.

1) Créez la classe `SortApplication`.
2) Ajoutez-y une méthode `static void displayArray(int[] array)` qui permettra de vérifier que le générateur fonctionne comme prévu. Pour ce faire, créez un tableau aléatoire et affichez-le sur la console.

Partie 2 – Tri par sélection

Vous allez maintenant trier les tableaux de données en utilisant l'algorithme de tri par sélection. Comme vu au cours, l'idée de ce tri est la suivante :



Explications :

a) Tri de `a[0]` :
   a. Phase 1 : on parcourt tous les éléments du tableau en partant de l'indice et on trouve le plus petit présent en stockant l'indice auquel il se trouve (ici 3).
   b. Phase 2 : On échange l'élément `a[0]` avec l'élément le plus petit (`a[3]`). `a[0]` contiendra ensuite l'élément le plus petit du tableau

b) Tri de `a[1]` :
   a. Phase 1 : on parcourt les éléments du tableau en partant de l'indice 1 et on trouve le plus petit élément présent en stockant l'indice auquel il se trouve (ici 2).
   b. Phase 2 : On échange `a[1]` avec `a[2]`. `a[1]` est désormais trié.

c) Tri de `a[2]` :
   a. Phase 1 : on parcourt le tableau en partant de l'indice 2 et on trouve le plus petit élément présent en stockant l'indice auquel il se trouve (ici 7).
   b. Phase 2 : On échange a[2] avec a[7]. a[2] est désormais trié.

d) …

On continue ainsi jusqu'à ce que tout le tableau soit trié.

Tâche 3

1) Implémentez une classe abstraite `SelectionSort` avec une méthode statique implémentant un tri par sélection. Le prototype de la méthode à implémenter est le suivant :
   `static public int[] sort(int[] array).`

2) Vérifiez le bon fonctionnement de votre méthode de tri en utilisant la méthode `displayArray` que vous avez implémenté à la tâche 2. Tous les éléments doivent maintenant être affichés dans l'ordre (ce qui était le but de la méthode de tri).

Tâche 4

Vous allez maintenant étudier l'efficacité de votre méthode de tri et la comparer avec une méthode de tri que nous vous fournissons. Pour mesurer le temps d'exécution d'une méthode, vous pouvez utiliser la méthode fournie par le système de Java, à savoir `System.currentTimeMillis()`. Cette méthode retourne un `long` contenant le nombre de millisecondes écoulées depuis minuit le premier janvier 1970. Vous pouvez également utiliser `System.nanoTime()` qui retourne le nombre de nanosecondes écoulées depuis un moment arbitraire. Cette méthode est plus précise et permet d'avoir une meilleure résolution de calcul.

En appelant cette méthode deux fois, il est possible de calculer le temps écoulé entre ces deux appels. Dans cette tâche, vous allez ainsi devoir mesurer et noter à l'aide de cette technique le temps d'exécution de votre tri pour les trois types de tableaux.

Pour vous assurer d'avoir des mesures significatives, il faut répéter chaque mesure 10 fois (sur des tableaux différents). Assurez-vous de faire ces mesures directement dans le code. Attention, vous ne devez mesurer que le temps de tri et non le temps de création du tableau ! Pour les mesures de temps, vous devez mesurer le temps de tri pour des tableaux de 10k à 100k éléments, par pas de 200 (càd 10000 éléments, puis 10200 éléments etc…). A chaque tri, vous devez créer un nouveau tableau à trier et vous devez répéter ces tris pour les trois types de tableaux possibles (aléatoire, inverse, mélangé), avec comme valeur pour `maxValue` la taille du tableau. Faites les graphes avec toutes les valeurs que vous avez mesuré. Notez également, comme résumé et sous forme textuelle, les valeurs de temps obtenues pour 10k éléments, 20k éléments … 100k éléments.

Tâche 5

Nous vous fournissons sur le site la classe `YSort` qui implémente un algorithme de tri plus compliqué mais que nous soupçonnons être plus rapide.

1) Testez cet algorithme de tri sur des tableaux de même taille que dans la tâche 4 (en faisant une moyenne sur 10 essais). Notez les résultats **sous forme textuelle** pour 10k éléments, 20k éléments, 30k éléments, … 100k éléments.
2) Faites un graphe (par exemple à l'aide d'Excel ou LibreOffice) sur lequel vous comparez graphiquement les résultats des deux méthodes de tri avec toutes les mesures que vous avez réalisées. Attentions notamment à libeller correctement les axes !
3) Est-ce que vous vous attendez à d'autres résultats en utilisant des tableaux créés avec `createShuffleArray` ou `createInvertedSortedArray` ? Indiquez votre raisonnement.

Tâche 6

Vous allez maintenant faire une analyse sommaire de la complexité du tri `YSort`.

1) Trouvez de manière **empirique** la complexité du tri `YSort`.
2) A partir de cette information de complexité, quel est le temps d'exécution que l'on peut prévoir pour 10^{12} éléments à trier à l'aide du tri `YSort` ? Expliquez votre raisonnement.
3) Quel sera le temps d'exécution du tri du même nombre d'éléments à l'aide de votre tri par sélection ? Expliquez.